Principle Component Analysis - En jämförande laboration mellan GHA, PAST och egenvektorer

David Andersson

February 4, 2016

Abstract

Syftet med rapporten är att redogöra begreppet principal component analysis (PCA), samt att redogöra för de olika metoder som finns att tillgå för numeriskt hitta dem. Metoderna ska sedan appliceras på ett vanligt komprimeringsproblem.

1 Introduktion

Den första delen kommer handla om att hitta PCA med tre olika metoder, nämnligen egenvektorer, Generalized Hebbian Algorithm (GHA) samt Projection approximation subspace tracking (PAST). Som utvärdering så kommer det finnas en jämförelse mellan metoderna. Den andra delen kommer att handla om att applicera dessa tre metoder för att komprimera en bild.

För att lösa problemen används Matlab. All teori kring rapporten kommer från boken Independent Component Analysis, skriven av Aapo Hyvärinen, Juha Karhunen och Erkki Oja [1].

Rapporten utgår ifrån en grundläggande förståelse för principle component analysis.

2 Metod

2.1 Hitta PCA komponenterna

För att generera data så används gaussiskt distribuerade slumptal med medelvärde noll samt variansen ett. Vi genererar då tre realiseringar med 5000 sampler i varje, och multiplicerar med mix-matris A för att få ut observerade signaler [x1, x2, x3].

$$A = \begin{bmatrix} 1 & 0 & 0 \\ -0.5 & 0.5 & 0 \\ 0.3 & 0.1 & 0.1 \end{bmatrix}$$

$e = randn(3, 5000)$

$x = Ae$

2.1.1 Egenvektorer

För att hitta PCA i closed form så används egenvektorerna. Först beräknas kovariansfunktionen (Alternativt korrelationsfunktionen då medelvärdet är noll). Detta görs genom: $(x * x')/size(x)$. Sedan hittas egenvektorerna och egenvärderna med matlab-funktionen "eig". Iden är att de största egenvärdernas egenvektorer är också de riktningar i datan som har störst varians. Genom att hitta vektorer med stor varians så hittar man också de riktningar som bär på mest information. Jag är inte säker på om det behövs då det verkar som om matlab tar han om det automatiskt, men jag sorterade också egenvektorerna efter storlek enligt teorin. Resultatet kan ses i figur 1.

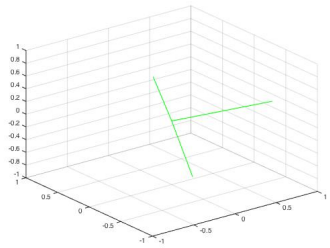

Figure 1: Egenvektorer

Figure 2: Resultat av komprimering med egenvektorer.

2.1.2 GHA

GHA är en stochastic gradient ascent algorithm (SGA), och en onlinemetod. Gradient descent betyder i korta drag att leta efter nollställen i en kostnadsfunktion genom att derivera i de olika riktningarna datan innehåller. Den negativa riktningen är dit algoritmen rör sig. Detta görs iterativt med en viss steglängd för att förhoppningsvis konvergera. GHA ortogonaliseras också genom att vid varje iteration subtrahera $tril(y * y')$, $y = w * x$, där tril extraherar alla värden under diagonalen. GHA kombinerar alltså en gradient (Oja's rule) med ortogonalisering (Gram-Schmidt).

När jag skulle försöka hitta en bra steglängd var det väldigt svårt, det närmaste jag kom var med 800 iterationer och 10^-3 som steglängd efter mycket laborerande med olika längd/iterations-kombinationer. Resultatet ses figur 3.

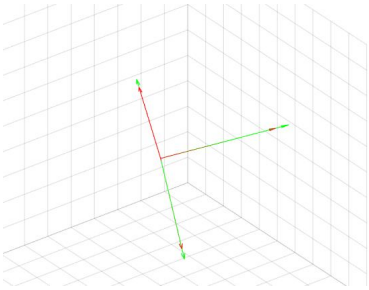

Figure 3: Vektorer från GHA (Egenvektorer i rött med längd 0.9).

Figure 4: Resultat av komprimering med GHA.

2.1.3 PAST

PAST är en recursive least square (RSL) algorithm och till skillnad från GHA så behövs ingen learning rate, alltså finns det ingen variabel som behövs justeras, vilket gör att den var enkel att implementera. Likt GHA ortogonaliseras algoritmen med tril. Även om handledningen specificerar tre vektorer så visas 2 i figur 5. Algoritmen ger oss inte principle components utan ortogonala vektorer i ett plan som spänns upp av de sökta egenvektorerna. Att söka efter tre av tre vektorer skulle då skapa en trivial lösning. Efter ca 20 iterationer konvergerar algoritmen och jag fick ganska konsekvent en viss vinkels fel men ändå ett rimligt resultat. Algoritmen ska enligt teorin konvergera snabbt.

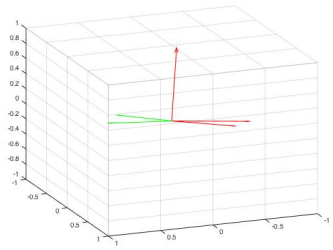

Figure 5: Vektorer från PAST (Egenvektorer i rött).

Figure 6: Resultat av komprimering med PAST.

3 Resultat

3.1 Egenvektorer

Eftersom egenvektorerna är optimala komponenter så använde jag projektionen på dem som referens. I figur 2 så syns komprimering med 10 egenvektorer.

3.2 GHA

Till skillnad från den tidigare uppgiften om GHA som krävde mycket liten steglängt och många iterationer för att konvergera, vilket den inte gjorde helt. I detta fallet så var det förvånansvärt smidigt att konvergera vid komprimering med ca 3 iterationer. Det hela tog ca 0.30 sekunder. Resultatet kan ses i figur 4. En annan sak var att jag fick ett väldigt bra resultat ifall jag uppdaterade W med en väldigt hög faktor. Bästa resultatet fick jag med att sätta alfa till mellan 0.1-10.

3.3 PAST

Min implementering av algoritmen krävde relativt många iterationer för att konvergera, jag räknade till ca 5 iterationer vilket tog ca 0.33 sekunder. Resultatet ses i figur 6. Beta satte jag till 1, då andra värden tenderade att divergera algoritmen, vilket inte är så konstigt då den är till för icke-stationära förändringar.

4 Slutsats

Angående de olika metoderna så är min uppfattning att om man utesluter beräkningstiden så var det mycket enklare att konvergera med PAST då den krävde få iterationer och det finns ingen steglängd att ta hänsyn till. I vissa fall, som till exempel i Independen component analysis, så kanske det inte räcker med att hitta ett underum, och då är såklart PAST ett problem. GHA fungerade också relativt bra, jag tror att med starkare konvergenskriterier så skulle jag fått ett bättre resultat. Egenvektorer är som bekant den optimala lösningen, men då inte resurserna finns att tillgå så är det potentiellt för långsamt.
Min jämförelse baserades alltså på kvaliteten av bilden i jämförelse med closed form versionen med egenvektorer. För att åstakomma likvärdigt resultat så tog det betydligt mycket mer tid med PAST. Troligtvis har det att göra med att det inte går att justera steglängden som i GHA. Det kan också vara så att min initiering inte var optimal, eller att mina konvergenskriterier inte var bra nog.
För att initiera GHA algoritmen så provade jag lite olika typer, men bäst resultat fick jag genom kompromissen att använda $eye(n) + rand(n)$ och sedan ortonormera. Utan ortonormeringen så var det svårare att kovergera vektorerna i uppgift 1 så de blev ortogonala. Och utan bruset tog det längre tid att konvergera bilden i uppgift 2. Det är svårt att se, men en liten skillnad kan ses i figur 8. För att initiera PAST så använde jag även där $eye(n)+rand(n)$, då algoritmen behövde en viss mängd brus för att nå ett bra resultat. Resultatet utan brus ses i figur 7.
Som förväntat så fanns det ett utbyte mellan att reglera steglängden som i GHA och ha en mer stationär algorithm som i PAST. Det spelade inte så stor roll hur många iterationer jag använde på PAST, utan resultatet var likvärdig. Det var till och med så att för många iterationer divergerade algoritmen. Jag fick

också ett konstant stationärt fel, som jag tänker kan hjälpas genom att optimera initieringen. Anledningen till att jag valde att jämföra algoritmerna med 10 egenvektorer var för att det var väldigt svårt att se skillnad på bilderna med över 50 vektorer.

Figure 7: Result av komprimering med PAST utan brus.

Figure 8: Result av komprimering med GHA utan brus.

References

[1] Hannu Oja and Klaus Nordhausen. Independent component analysis. *Encyclopedia of Environmetrics*, 2001.